

REMARKS

This Amendment is in response to the first Office Action dated November 21, 2002.

Claims 1-24 were examined in the Office Action. Claims 1-24 were rejected. Claims 2, 3, 7, 9-12, and 16 have been amended, as indicated in the enclosure entitled "Amendment Mark-Up." Claim 1 has been canceled. Reexamination and reconsideration are respectfully requested.

Drawing Objections

The drawings were objected to under 37 CFR 1.83(a) and also under 37 CFR 1.84(p)(5).

With respect to the drawing changes requested in Fig. 1, the Examiner noted the metadata module and the code module claimed in claim 1 were not shown in Fig. 1. The requested change to Fig. 1, therefore, adds these modules, as shown in red ink. This drawing change is supported in the original specification at page 9, lines 5-17 stating in part: "the output of front end 22 is illustrated as being a common language file 26 containing both metadata 28 and executable instructions 30. . . . The metadata portion 28 is produced by a metadata module within the front end system 22 and the executable instructions 30 are produced by a code module within the front end system." The sentence has been amended above to include reference numerals for both the metadata module and the code module. Given the support for this change in the original specification, no new matter has been added.

With respect to the drawing changes requested for Fig. 3, the Examiner noted that runtime environment claimed in claims 5 and 6 recited elements not shown in the Figures, specifically, a loader, a layout engine, a stack walker and a garbage collector. Consequently, these elements have been added to the runtime or execution environment shown in Fig. 3. This

drawing change is supported in the original specification at page 13, line 22 to page 14, lines 12, stating in part:

“Additionally, in the case where the execution environment 74 is a managed runtime environment, it may provide a number of other services during program execution. As an example, the system may provide a loader, which receives the executable file and resolves necessary references and loads the code. The environment may provide a stack walker, i.e., the piece of code that manages the method calls and provides for the identification of the sequence of method calls on a stack at a given point in time. A layout engine may also be provided, which establishes the layout, in memory, of the various objects and other elements as part of the application to be executed. The execution environment may further provide security measures to prevent unauthorized use of resources and other developer services, such as debuggers and profiling. Other types of services that can be provided by a managed execution environment include verification of code before it is executed, security facilities to determine whether certain code has permission to access certain system resources (or even execute at all), and memory management services, such as garbage collection. (Emphasis added here)

The paragraph has been amended above to include reference numerals for these elements. Given the support for this change in the original specification, no new matter has been added.

With respect to the requested change to Fig. 4, the reference numeral “96” has been removed as it is not referenced in the original specification. It is believed that this change to Fig. 4 should obviate the drawing objection to Fig. 4 under 37 C.F.R. 1.84(p)(5).

With respect to the requested change to Fig. 5, the reference numeral “100” was inadvertently omitted from the original drawing and has been added in red ink. Support for this change is found in the original specification at page 16 describing an “exemplary system 100”. No new matter has been added.

Regarding the objection to Fig. 7 noting that Item 322 was not mentioned in the specification, Applicant notes that the paragraph beginning at page 7, line 5 of the original specification has been amended above to refer to item 322. As a result, it is believed that this amendment should obviate the drawing objection to Fig. 7 under 37 C.F.R. 1.84(p)(5).

Claim Rejections – 35 U.S.C. § 101

Claims 12 and 16 were rejected under 35 U.S.C. § 101 because the claimed invention is directed to non-statutory subject matter. Both claims 12 and 16 have been amended above in the form dictated by In re Lowry, 32 F.3d 1579, 32 USPQ2d 1031 (Fed. Cir. 1994). It is believed that claims 12 and 16, as amended, recite statutory subject matter under 35 U.S.C. § 101 and reconsideration of this rejection is respectfully requested.

Claim Amendments: Claims 2 and 3

Claims 2 and 3 originally depended from claim 1. Claim 1 has been canceled herein such that claims 2 and 3 have been amended to include the limitations of independent claim 1. The amendments were made to maintain the claims pending in light of canceling claim 1 and thus were not made for patentability purposes. Moreover, the scope of claims 2 and 3 has not changed by this amendment.

Claim Rejections – 35 U.S.C. § 112

Claims 2-6, and 10-24 were rejected under 35 U.S.C. § 112, first paragraph, as containing subject matter which was not described in the specification in such a way as to enable one skilled in the art to which it pertains, or with which it is most nearly connected; to make and/or use the invention. As this rejection is understood, these claims are rejected because one of ordinary skill in the art would be unable reproduce or replicate “consuming” or “compiling” or “re-compiling” a “common language file” or “common language library” because two possible interpretations exist for such “consuming” or “compiling” or “re-compiling”, wherein the interpretations are opposing. The Examiner provides the two possible interpretations follows:

- a. The common language file or library metadata and executable code is separated from the included/imported common language file or library and incorporated into the new second common language file which is produced from a second native source code file; or
- b. The common language file or library metadata is incorporated into the second-complied common language file, as produced by the compilation of the second native source code file, by means of reference to the data members and function/methods of the first-complied common language file or library but without incorporating the executable code of the first-compiled common language file or library into the second-compiled common language file. By this means, at runtime, the second-compiled common language file loads into memory the first-compiled common language file or library and references the data members and methods/functions of the first-compiled common language file external to the internal operation of the second-compiled common language file.

The Examiner further states that if the first of the two interpretations is correct, then Fig. 2 is correct but if the second of the two interpretations identified above is the correct interpretation, then Fig. 2 is incorrect. Applicants respectfully disagree. Indeed, the application is not limited to one interpretation or the other, and regardless of which interpretation should be chosen, Fig. 2 correctly illustrates each.

Referring to the original specification, it is clear that various embodiments of the present invention may incorporate aspects relating to a front end system that consumes metadata information and native source code. The specification describes importing a common language

file, parse or analyze the metadata and build a symbol table accordingly. As stated on page 9 of the specification, the front end system of the present invention “may read or receive either metadata, executable instructions or a combination of both.” For a front end system to simply read and/or receive executable instructions, the front end system would have to produce an output file that incorporates the instructions. That is, without some other reference information, it is clear that this information would be incorporated as the Examiner noted with respect to the first interpretation above. Thus, the first interpretation has been disclosed and enabled. Since Fig. 2 reflects this embodiment, wherein the intermediate file 50 incorporates information from the common language file, Fig. 2 is accurate.

As stated with respect to another embodiment “the front end should be able to parse the common language file and convert the type and method information in the metadata into proper form for the particular symbol table. For each new class that is added to the symbol table, the common language file must be read for the methods that are supported by the new class.” (Specification, page 24, lines 7-11; See also Figs. 6 and 7.) Given the extensive description of parsing of the metadata and building a symbol tables, in conjunction with Figs. 6 and 7, it is believed that the process of consuming a common language file in combination with native source code is clearly enabled to build a symbol table. The creation of a new symbol table from the metadata is one method of building an intermediate language file by reference to the data members and function/methods of the first-complied common language file or library but without incorporating the executable code of the first-compiled common language file or library into the second-compiled common language file. Consequently, the second interpretation described above is enabled in the present application. Moreover, since Fig. 2 is designed to show that the consumption of a common language file in conjunction with a second file, e.g.,

native language source file 48 to produce a new file, e.g., intermediate file 50 it is not inaccurate with respect to this second interpretation. Instead, it does show the produced intermediate language file and that this file is subsequently loaded into an execution environment. Further, Fig. 3 demonstrates the loading of multiple files that may allow or provide reference functionality. Taken together, Fig. 2 is accurate with respect to the second interpretation provided above.

Given this, it is believed that those skilled in the art will recognize Fig. 2 as illustrating the first interpretation as well as the second. Therefore, reconsideration of the rejections under 35 U.S.C. §112 is respectfully requested.

Claim Rejections – 35 U.S.C. § 102

Claims 1, 7-9, and 12-15 were rejected under 35 U.S.C. § 102(a) as being anticipated by Blickstein (USPN 5,577,253).

Claim 1 has been canceled above such that the rejection related to claim 1 has been obviated.

With respect to the remaining claims, the Applicant respectfully traverses the Examiner's rejections based on the Blickstein reference cited above. That is, the cited reference does not identically disclose all of the limitations of the claimed invention. More specifically, with respect to claim 7, as amended, Blickstein does not identically disclose "a common language instructions section having instructions in a common language, the instructions related to the written program functions of the native source code file and consumed metadata." Similarly, with respect to claim 12, Blickstein does not identically disclose an executable instructions section that is suitable for use with a plurality of source languages.

Under 35 U.S.C. § 102, a reference must show or describe each and every element claimed in order to anticipate the claims. *Verdegaal Bros. v. Union Oil Co. of California* 814 F.2d 628 (Fed. Cir. 1987) ("A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference.").

The Office Action states Blickstein "generates from them an intermediate language representation of the program expressed in the source code" and that the intermediate language is "constructed to represent ... source code languages in a [sic] universal manner, so the interface between the front end and the back end is of a standard format." (Office Action, Page 7). The intermediate language described in Blickstein appears to provide a generic or common language that can be used to represent functions originating in different native source code languages. However, Blickstein must have different front ends "tailored" for each different source language. (Blickstein, Col. 3. line 7.) Although Blickstein provides a relatively generic interface, each front end can only consume or compile a single language. Consequently, the instructions section of a single instance of a common language file of Blickstein cannot simultaneously reflect the content of written functions of multiple source languages. That is, each intermediate language file generated by can only represent the content of one native source language.

Since Blickstein does not disclose the use of an instruction section of a file for representing instructions originally represented in different languages, Blickstein cannot, as a matter of law, anticipate claims 7 and 12. Further, claims 8-9 depend directly from claim 7 and claims 13-15 depend directly from claim 12 such that claims 8-9 and 13-15 should also be allowed over Blickstein.



Conclusion

As originally filed, the present application included 24 claims, 7 of which were independent. As amended, the present application now includes 23 claims, 8 of which are independent. Accordingly, a check for \$ 84.00, for the 1 additional independent claims exceeding 7 independent claims is submitted herewith. It is believed that no further fees are due with this Response. However, the Commissioner is hereby authorized to charge any deficiencies or credit any overpayment with respect to this patent application to deposit account number 13-2725..

In light of the above remarks and amendments, it is believed that the application is now in condition for allowance, and such action is respectfully requested. Should any additional issues need to be resolved, the Examiner is requested to telephone the undersigned to attempt to resolve those issues.

Respectfully submitted,

A handwritten signature in black ink, appearing to read "T. B. Scull".

Timothy B. Scull, Reg. No. 42,137
MERCHANT & GOULD P.C.
P.O. Box 2903
Minneapolis, MN 55402-0903

Dated: 3/21/03



RECEIVED

MAR 28 2003

Technology Center 2100

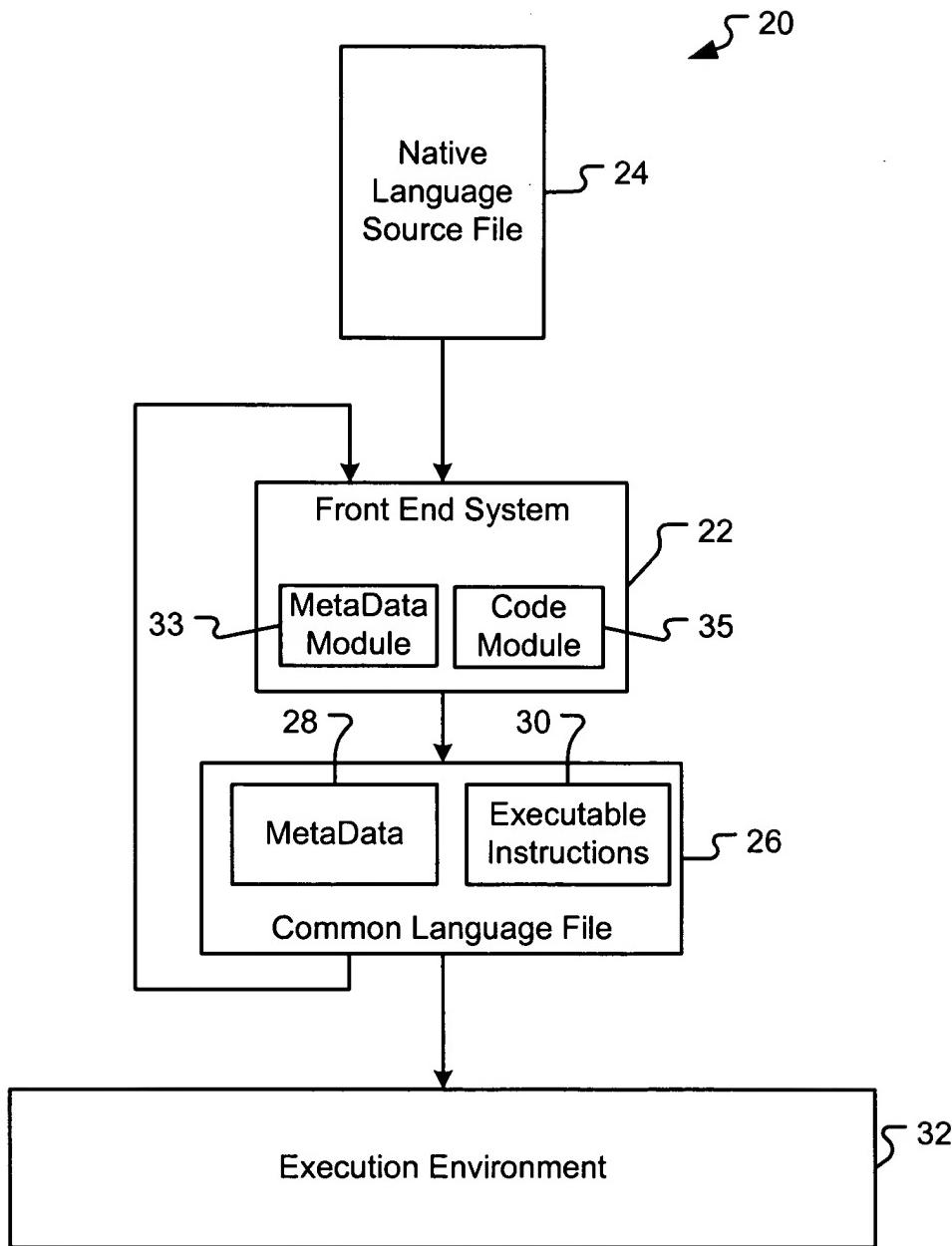


Fig. 1

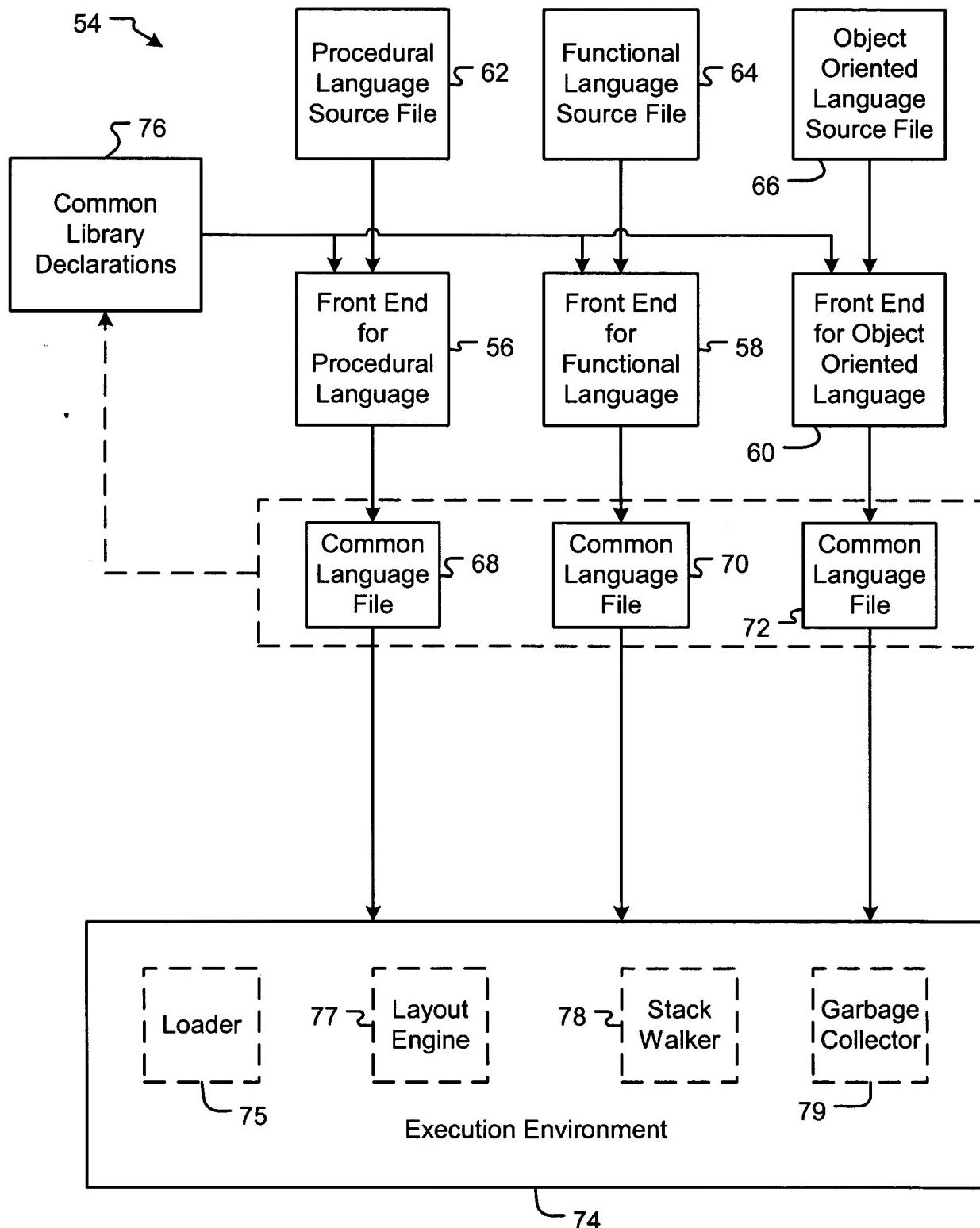


Fig. 3



80

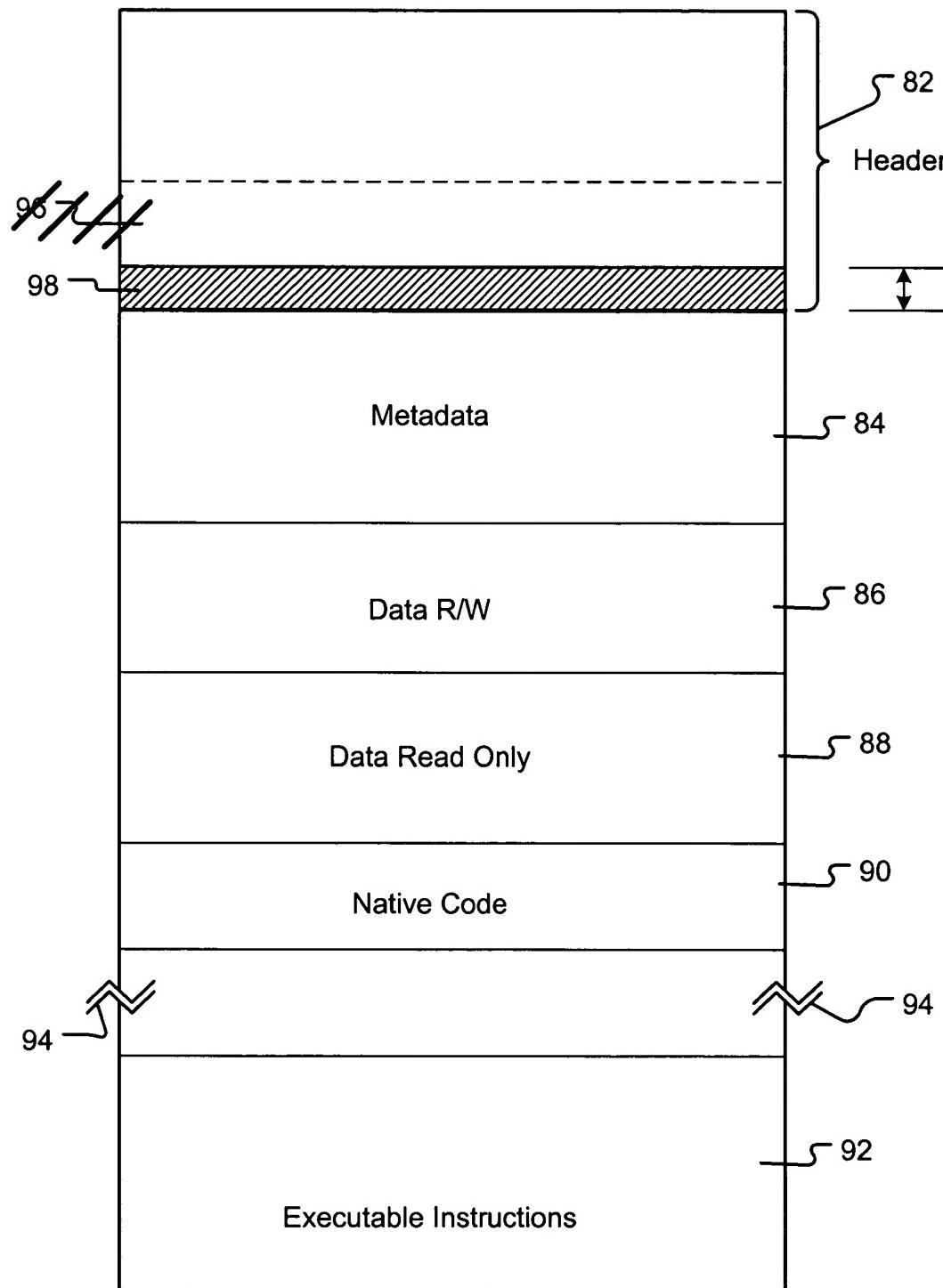


Fig. 4

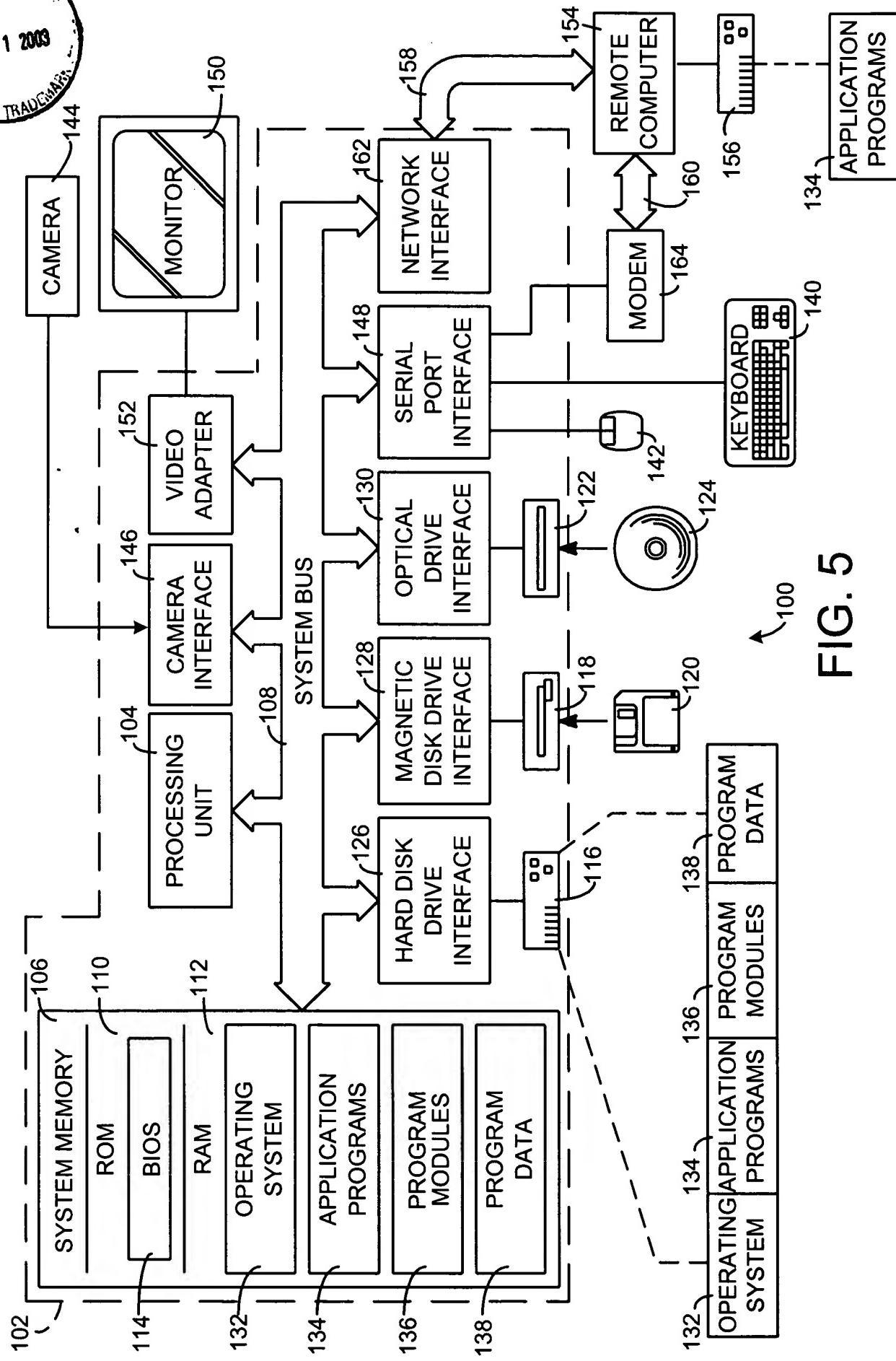


FIG. 5